



Processing

Introduzione a Processing

di MATTEO FIORAVANTI

Scopriamo l'Object Oriented Programming, che rappresenta un modo più semplice di costruire le nostre interfacce. Quarta puntata.

Nella scorsa puntata abbiamo imparato a disegnare dei controlli grafici: indicatori, pulsanti, LED, manopole. Per vedere all'opera tali elementi, abbiamo costruito una semplice interfaccia grafica per un programma che controlla Arduino. In questa quarta puntata risolveremo la problematica di dover costruire interfacce grafiche complesse che richiedano, magari,

l'uso di molti controlli e indicatori. Infatti l'interfaccia costruita nella terza puntata funzionava come un unico programma, cioè il codice relativo alla gestione dei controlli e indicatori era fuso all'interno del codice principale del programma, il quale controllava completamente l'interfaccia. In questa sede risolveremo altresì il problema di richiamare all'interno del



Fig. 1

programma principale i veri LED, interruttori, manopole, nelle quantità che ci occorrono, senza preoccuparci del codice occorrente ad essi, ma interessandoci unicamente al loro comportamento in termini di ingressi ed uscite (come del resto accadrebbe con i controlli reali).

Conosciamo già l'uso delle *funzioni* in Processing; ricordiamo che una funzione è definita all'interno del programma, viene richiamata a seconda dei casi e ne gestiamo ingressi ed uscite. Per il problema che affrontiamo, ci serve qualcosa di simile. Supponiamo di dover inserire nella nostra interfaccia tre LED, ognuno dei quali avrà un suo significato e sarà collegato a certe variabili, proprio come accadrebbe in un sistema reale.

Alla luce di quanto spiegato nella scorsa puntata, la prima cosa che ci viene in mente è piazzare tre immagini dei LED e modificarle a seconda delle variabili che dovrebbero rappresentare.

L'interfaccia siffatta sarà certamente funzionante, ma il relativo codice potrebbe diventare ingestibile nel momento in cui ci trovassimo a dover aggiungere altri LED ai precedenti, oppure laddove dovessimo essere costretti a complicare l'architettura del programma.

Pensiamo invece a quanto sarebbe più comodo se potessimo richiamare una specie di *costruttore* di LED, che ci permettesse di costruire i componenti led1, led2 led3 (cioè i nostri tre LED virtuali) e così via, e se ogni diodo luminoso potesse essere gestito in completa autonomia, cioè avesse un codice a sè stante e direttamente controllabile.

Potremmo, infatti, attribuire lo stato di *Acceso* a led1, mentre cambiamo led2 da *Acceso* a *Spento* e così via.

Se avessimo questa possibilità, ci basterebbe concentrarci unicamente sul programma principale ed ogni LED si com-

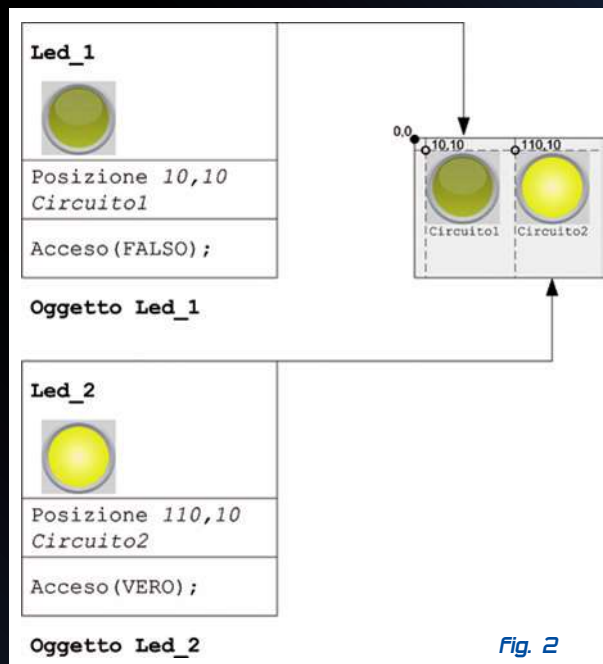


Fig. 2

porterebbe proprio come se fosse reale. Fare questo è possibile, ma dobbiamo introdurre una tecnica di programmazione nuova e che risolva in maniera estremamente semplice questi casi; tale tecnica si chiama programmazione orientata agli oggetti, ovvero, abbreviando, OOP (acronimo di Object Oriented Programming). Processing permette di implementare questa tecnica un modo del tutto naturale. Per spiegare l'OOP partiamo introducendo il concetto di Classe e di Oggetto, poi trasformeremo i controlli descritti nella lezione precedente a loro volta in Classi ed Oggetti e, con quanto appreso, scriveremo il codice di un'interfaccia che simulerà un sistema reale.

IL CODICE DI QUESTA PUNTATA

Qui di seguito elenchiamo i programmi di esempio di questa puntata del corso.

1.LEZIONE 4_1 - INTRODUZIONE ALLA PROGRAMMAZIONE AD OGGETTI.

Questo codice permette di costruire un'interfaccia con tre oggetti appartenenti alla Classe Led; serve, inoltre, per familiarizzare con i concetti di Classe e di Oggetto.

2.LEZIONE 4_2 - INTERFACCIA GRAFICA COMPLESSA. Questo codice permette di costruire un'interfaccia grafica complessa, nella quale tutti i controlli ed indicatori della terza lezione sono diventati Classi ed Oggetti.

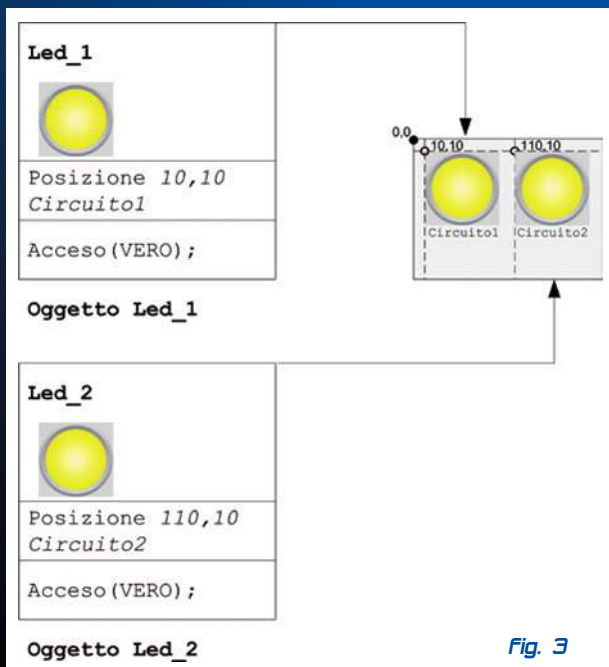


Fig. 3

LEZIONE 4_1 - INTRODUZIONE ALLA PROGRAMMAZIONE AD OGGETTI

Partiamo ora da una minima astrazione, definendo il LED come una *Classe*; ciò significa innanzitutto che alla *Classe* dobbiamo dare un *Nome*, quindi bisogna definire dei *Campi* e dei *Metodi*. Tutto questo è rappresentato in Fig. 1.

Il *Nome* della *Classe* è quello che identificherà l'*Oggetto* da essa costruito, mentre i *Campi* sono gli attributi (ad esempio la posizione X ed Y dell'oggetto e l'etichetta) mentre i *Metodi* sono i comportamenti che possono essere eseguiti. Nel caso del LED, abbiamo definito come metodo **Acceso()**: in pratica, **Acceso(true)** accenderà il diodo luminoso, mentre **Acceso(false)** lo spegnerà. Quindi, i *Campi* di una *Classe* sono degli attributi, mentre i *Metodi* sono le azioni, ovvero i comportamenti che possono essere intrapresi.

La *Classe* è un insieme generico; di essa si definiscono i singoli *Oggetti*, come ci mostra la Fig. 2. Infatti, se definiamo come *Nome* **Led_1**, come *Posizione* 10,10 e per *Etichetta* **Circuito1**, avremo definito un *Oggetto* che si chiama **Led_1**, appartiene alla *Classe* **Led**, si trova nella posizione 10,10 e mostra come etichetta "Circuito1". Ora, se volessimo accendere questo oggetto, dovremmo intervenire sul metodo **Acceso**, scrivendo quanto segue:

```
Led_1.Acceso(true);
```

Se lo volessimo spegnere, opereremmo sempre attraverso lo stesso metodo, ma scrivendo:

```
Led_1.Acceso(false);
```

Volendo inserire nel nostro ipotetico programma un secondo LED, basterà definire un *Oggetto* che, verosimilmente, chiameremo **Led_2**, sempre appartenente alla *Classe* **Led**, ma avente attributi differenti, una diversa posizione X ed Y (110,10) ed una differente Etichetta (**Circuito2**). Quindi abbiamo creato, o meglio Processing ha costruito, un nuovo *Oggetto* chiamato **Led_2**, diverso da **Led_1**, ma avente comunque gli stessi *Metodi*. Infatti, se volessimo accendere **Led_2** opereremmo in questo modo:

```
Led_2.Acceso(true);
```

Supponiamo, ora, di avere all'interno del programma la situazione rappresentata in Fig. 2, ossia **Led_1** spento e **Led_2** acceso. Operando in termini di *Metodi*, possiamo accendere **Led_1** e passare alla situazione di Fig. 3, scrivendo:

```
Led_1.Acceso(true)
```

Chiaramente quanto riportato fin qui è un esempio che dovrebbe aiutarci a capire il concetto di *Classe* ed *Oggetto*.

Scendiamo ora sempre più sul caso pratico, per vedere come tutto questo si applica. Consideriamo il codice di esempio (Lezione_4_1.pde): prima di tutto gli *Oggetti* devono essere dichiarati, proprio come succede con le variabili; infatti nel nostro codice abbiamo bisogno di tre LED (**ld1**, **ld2**, **ld3**), quindi dichiareremo tre oggetti "led" appartenenti alla *Classe* **Led**, come riportato in Fig. 4. Poi, dentro **void setup()** andremo a fare ciò che prende il nome di istanza degli oggetti precedentemente dichiarati; in pratica, ciò si fa attraverso il comando **new**. Perciò, la nuova istanza dell'oggetto **ld1** assumerà questo aspetto:

```
ld1 = new Led(10, 10, "Led 1");
```

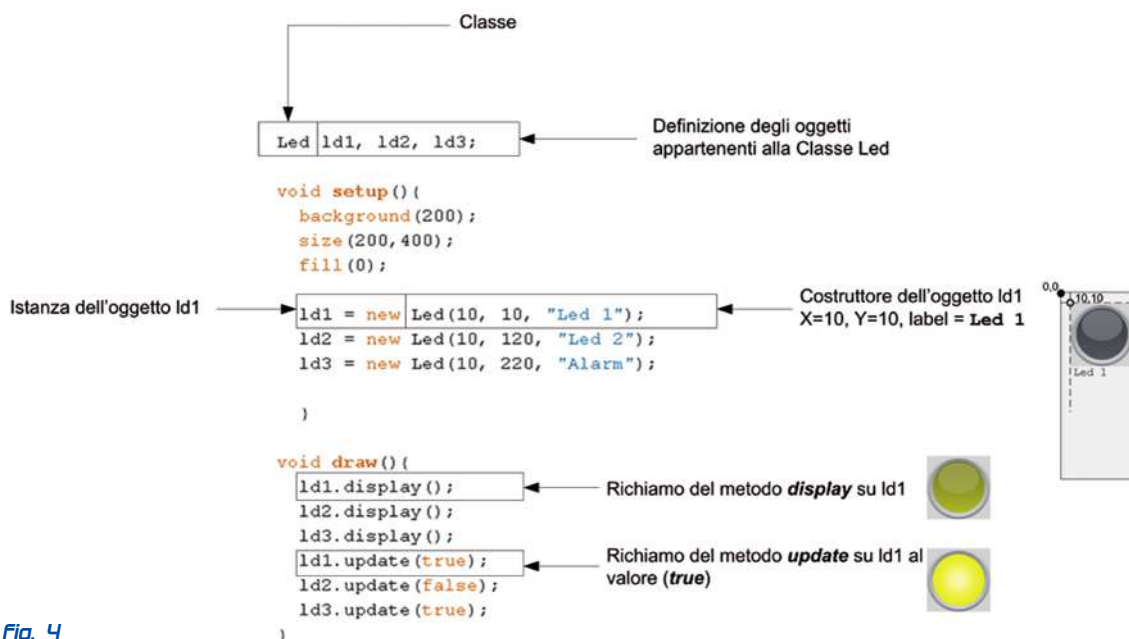



Fig. 4

Ciò significa che il nuovo oggetto `ld1` della *Classe* `Led` deve essere costruito con i seguenti campi: posizione `X = 10`, posizione `Y = 10`, Etichetta = `"Led 1"`. Procediamo allo stesso modo per gli altri *Oggetti*, come riportato in Fig. 4. Dentro il ciclo principale `void draw()`, si andrà ad agire sui *Metodi* di `Led`. Per la *Classe* `Led` abbiamo definito due *Metodi*: `.display()` e `.update()`. Il primo fa letteralmente apparire il LED (ciò significa che se non viene richiamato questo metodo, il diodo luminoso non compare nell'interfaccia) anche se è stato definito ed istanziato, mentre il secondo consente

di aggiornare dinamicamente lo stato del LED, facendolo passare da acceso a spento e viceversa.

Il codice del programma corrispondente è riportato di seguito nel Listato 1 ed è anche visibile in Fig. 4.

A questo punto abbiamo visto come si opera con le *Classi* e con gli *Oggetti*; rimane da capire come questi si programmano, ed è quanto faremo ora. Scrivere il codice per definire una *Classe* non è per nulla differente dallo scrivere codice per un programma o per una funzione; cambiano solo alcuni dettagli.

Prima di tutto la *Classe* (che ad esempio

Listato 1

```

Led ld1, ld2, ld3; //dichiarazione degli oggetti ld1, ld2, ld3 appartenenti alla Classe Led

void setup(){ //inizializzazione
  background(200); //sfondo
  size(200,400); //dimensione dell'area grafica del programma
  fill(0);

  ld1 = new Led(10, 10, "Led 1"); //istanza dell'oggetto ld1 e sua costruzione
  ld2 = new Led(10, 120, "Led 2"); //istanza dell'oggetto ld2 e sua costruzione
  ld3 = new Led(10, 230, "Alarm"); //istanza dell'oggetto ld3 e sua costruzione
}

void draw(){
  ld1.display(); //richiamo del metodo .display() dell'oggetto ld1, in pratica ld1 viene visualizzato
  ld2.display(); //identica cosa per ld2
  ld3.display(); //identica cosa per ld3
  ld1.update(true); //richiamo del metodo .update() per ld1, mettendolo a "true" si accende il led ld1,
                  //mettendolo a "false" si spegne
  ld2.update(false); //ld2 viene spento
  ld3.update(true); //ld3 viene acceso
}

```

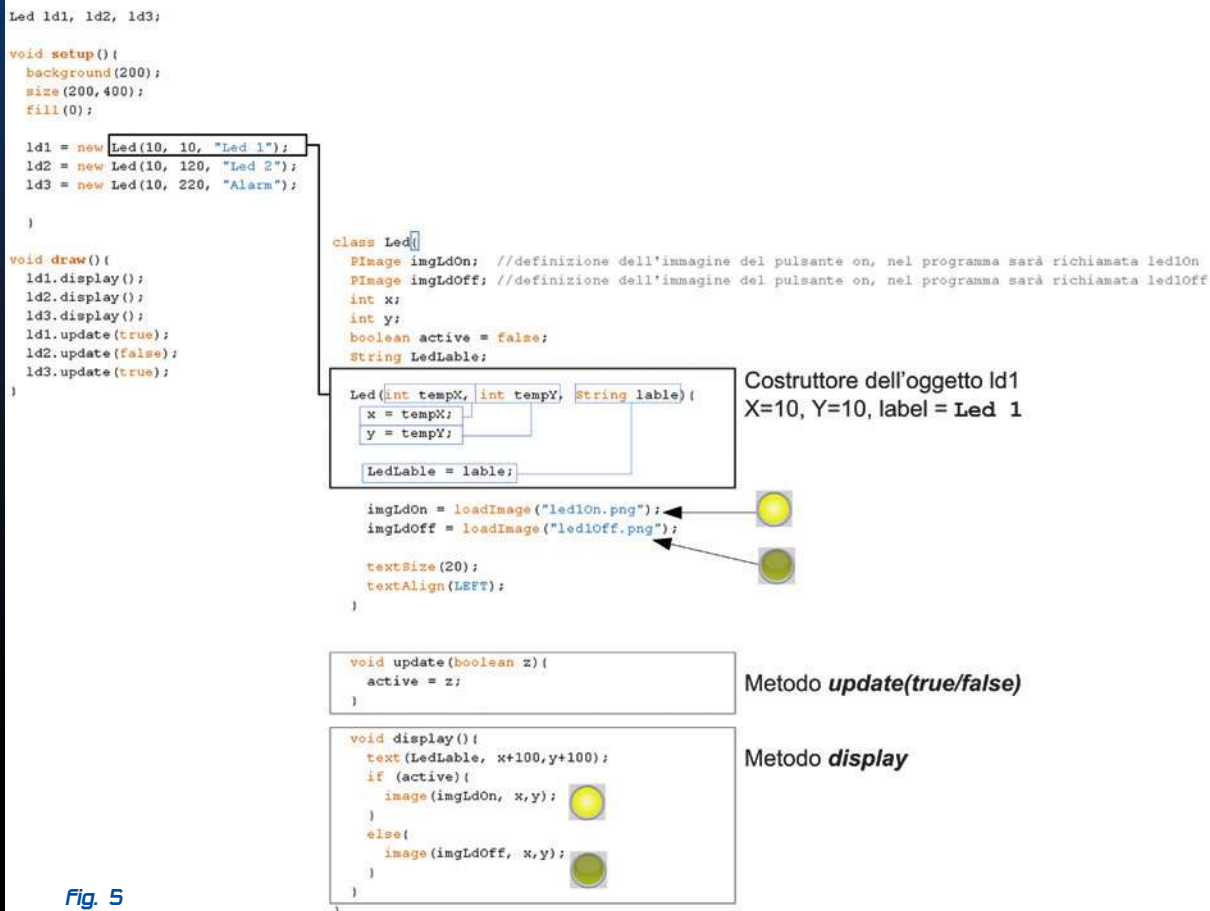


Fig. 5

può essere Led) è definita all'interno di due parentesi graffe:

```
class Led{}
```

Questo significa che tutto quanto serve alla *Classe* Led per funzionare è contenuto all'interno di tali parentesi.

Ci saranno delle variabili locali e inoltre verranno definiti il costruttore dell'interfaccia e i metodi. Nel Listato 2 è riportato il codice commentato per la *Classe* Led e lo stesso è visibile nella Fig. 5, che mostra come questo codice interagisce con il programma principale.

Un aspetto importante su cui porre l'attenzione è che la classe, per funzionare, potrebbe necessitare di sue variabili interne; tali variabili sono definite come in qualsiasi altro programma di Processing e soprattutto sono delle variabili locali. Quando viene istanziato un nuovo oggetto, abbiamo visto come questo debba essere definito nei suoi *Campi*; questi ultimi saranno poi passati alle variabili locali della *Classe*, così come mostrato dalla Fig.

5, attraverso quello che viene definito il *Costruttore dell'Oggetto*. Per la *Classe* in questione, quest'ultimo assume la seguente forma:

```
Led(int tempX, int tempY, String label) {}
```

Il *Costruttore* è in definitiva il ponte che consente di trasferire i *Campi*, definiti quando l'*Oggetto* viene istanziato, alle variabili locali che servono poi al programma per costruire l'*Oggetto*.

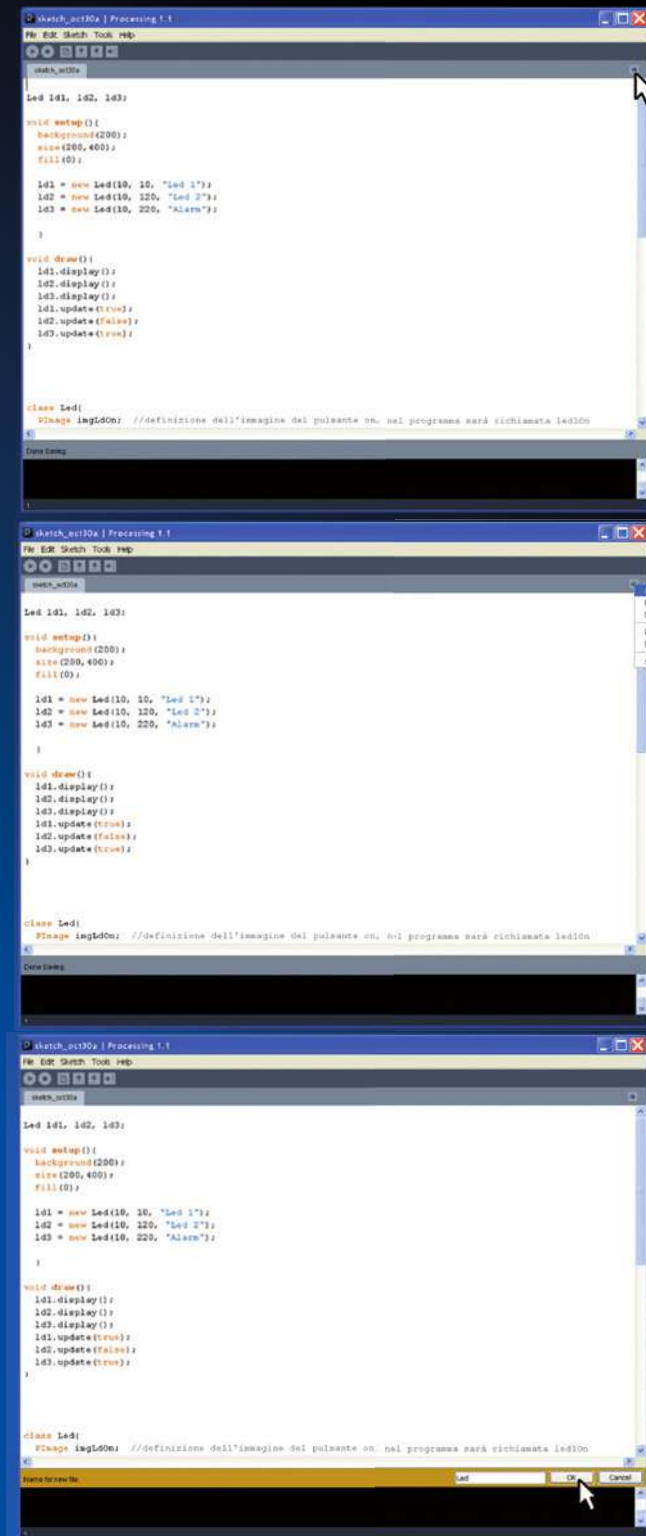
Riassumendo, i *Campi* vengono definiti nel programma principale quando si istanzia il nuovo *Oggetto*; nella *Classe*, i *Campi* sono trasferiti alle variabili locali attraverso il *Costruttore dell'Oggetto*.

La Fig. 5 rappresenta anche, in maniera schematica, i due *Metodi*, i quali all'interno della *Classe* sono sostanzialmente delle funzioni; ad esempio:

```
void display() {}
```

Quindi il *Metodo*, in realtà, è una funzione, che però viene richiamata dall'*Oggetto*

Fig. 6



se *Led*) la parte riguardante la *Classe* può essere scritta in fondo al programma principale, cioè praticamente fuori da `void draw()`, analogamente a quanto si fa per le funzioni. Però esiste un modo che ci consente di rendere il codice ancora più leggibile: si tratta di sfruttare una funzionalità dell'editor di Processing, che è quella che ci permette di aprire dei nuovi Tab di editor e che ancora non abbiamo affrontato. La Fig. 6 ci illustra come ciò viene fatto e comunque la procedura è la seguente: innanzi tutto occorre andare con il cursore, nell'angolo in alto a destra dell'editor, sopra alla rappresentazione della freccia verso destra, e fare clic con il tasto sinistro del mouse. Allora

si aprirà un menu a tendina e bisognerà selezionare **New Tab**; a questo punto comparirà una barra gialla sotto l'editor, contenente la richiesta di inserimento del nome per il nuovo file "New name for file". Nella casella di testo bisogna scrivere il nome del nuovo Tab, che noi con "grande originalità" chiameremo *Led*. Di fianco al Tab indicante il nome del file, ne sarà comparso un altro con il nome di "Led"; facendovi clic, si aprirà una nuova pagina di editor. Ora non rimane altro da fare che copiare ed incollare tutto il codice relativo alla *Classe Led*, che avevamo scritto nel programma principale, ed incollarla nel nuovo Tab *Led* (Fig. 7). A questo corrisponde effettivamente un nuovo file, che si trova nella stessa cartella del programma principale. Il codice, quindi, rimane semplicemente quello di Fig. 8, ma soprattutto, la *Classe Led* che abbiamo creato possiamo trasportarla in altri programmi, spostando il file *Led.pde* nelle cartelle dei programmi che andremo a creare; inoltre, dentro ai programmi potremo creare tutti gli oggetti "led" che vorremo, ricordandoci di definirli all'inizio del programma, di istanziarli e di usarli con i relativi metodi. Questa tecnica consente di semplificare notevolmente il codice e di riutilizzarlo tutte le volte che vorremo nei programmi che andremo a creare. Dal punto di vista della creazione di interfacce grafiche, questo permette, una volta che

proprietario, nel programma principale:

```
...
ld1.display();
...
```

Nel Listato 2 (codice riguardante la clas-

Listato 2

```

class Led{
  PImage imgLdOn; //definizione dell'immagine del pulsante on, nel programma sarà richiamata led10n
  PImage imgLdOff; //definizione dell'immagine del pulsante on, nel programma sarà richiamata led10ff
  //dichiarazioni delle variabili funzionali all'esecuzione della classe
  int x;
  int y;
  boolean active = false;
  String LedLabel;

  //costruttore
  Led(int tempX, int tempY, String lable){ //la Classe Led necessita di tre campi che devono essere definiti
                                          // quando si istanzia un nuovo oggetto
                                          //la posizione X, Y ed una stringa che definisce l'etichetta
    x = tempX; //il campo tempX viene passato alla variabile globale della Classe X
    y = tempY; //il campo tempY viene passato alla variabile globale della Classe Y

    LedLabel = lable; //il campo lable viene passato alla variabile globale della Classe LedLabel

    imgLdOn = loadImage("led10n.png");//vengono caricate le immagini .png rappresentative del led acceso e
    spento
    imgLdOff = loadImage("led10ff.png");

    textSize(20); //viene definita dimensione del carattere
    textAlign(LEFT); //viene definito l'allineamento del testo, a sinistra
                    //questi parametri servono per la rappresentazione dell'etichetta
  }

  //METODO update() -----
  void update(boolean z){ //il metodo si aspetta che sia passato un parametro true/false
    active = z; //il valore del parametro è trasferito alla variabile globale "active", della Classe
  }
  //-----

  //METODO display() -----
  void display(){
    text(LedLabel, x+100,y+100);
    if (active){ // a seconda del valore della variabile "active" il metodo display() mostra un'immagine
      diversa del led
      image(imgLdOn, x,y); //immagine del led acceso, se active == true
    }
    else{
      image(imgLdOff, x,y); //immagine del led spento, se active == false
    }
  }
  //-----
}

```

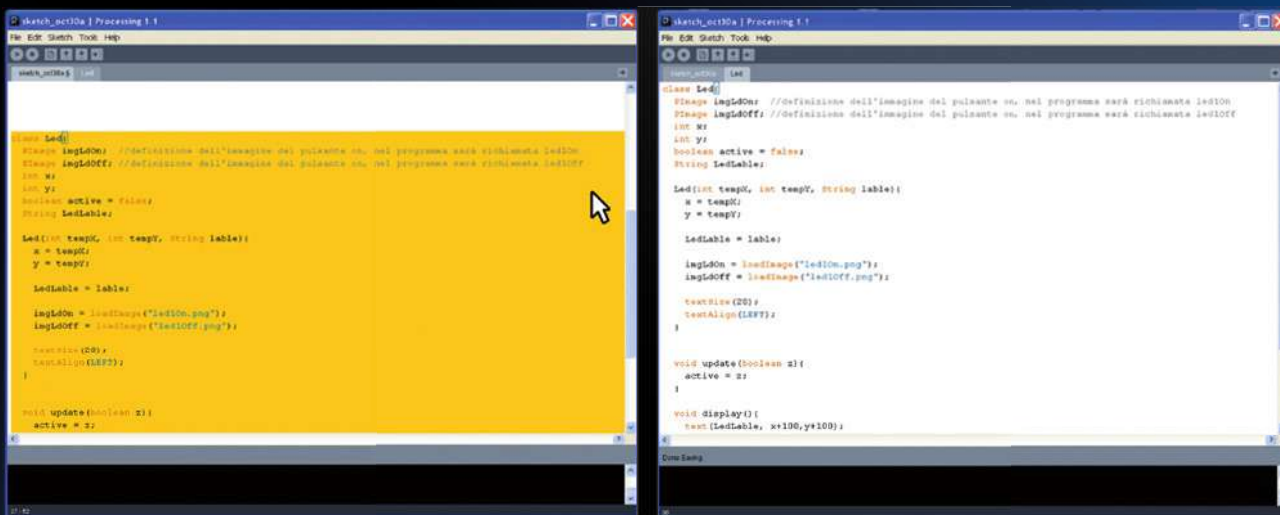
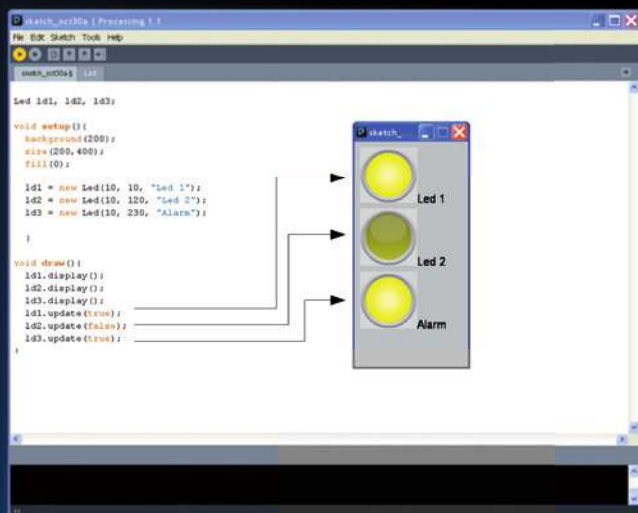


Fig. 7



avremo creato i nostri controlli, di concentrarci unicamente sul programma.

LEZIONE 4_2 - INTERFACCIA GRAFICA COMPLESSA

Il secondo codice presentato in questa lezione ci serve per prendere dimestichezza con i controlli di tipo Led, Knob, Button ed Indicator, trasformati però in Classi e quindi in *Oggetti*. Supponiamo di voler realizzare l'interfaccia di Fig. 9, dove si trovano tre interruttori, tre LED, una manopola e un indicatore analogico. Stabiliamo a questo punto una funzionalità puramente di esempio: l'interruttore denominato "Start" attiva un ipotetico sistema che incrementa la sua velocità e questa viene indicata dalla lancetta dell'indicatore analogico; il livello di questa velocità è settato tramite la manopola "Level" e quando raggiunge una soglia, si deve accendere il LED denominato "Alarm". Questa funzionalità è

Fig. 8

rappresentata in Fig. 10. Aggiungiamo un'ulteriore complessità, cioè facciamo in modo che i due interruttori siglati "Circuit 1" e "Circuit 2" attivino, rispettivamente, "Led 1" e "Led 2" (Fig. 11). L'interfaccia qui rappresentata è in definitiva un simulatore: non serve a nulla di reale, ma ci consente di apprendere una maniera più efficace di programmare questo genere di applicazioni. Se vogliamo che la nostra interfaccia controlli realmente qualcosa, possiamo riprendere le tecniche illustrate nella lezione precedente e magari collegare il tutto ad Arduino. Il codice contenuto nella cartella Lezione_4_2 contiene sia il programma principale che simula l'interfaccia, sia tutte le classi usate in questo esempio: Led, Button, Indicator e Knob. Il codice del programma è strutturato come nell'esempio precedente, quindi nella fase iniziale in cui si dichiarano gli oggetti, nella fase di istanza degli oggetti e nel vero e proprio ciclo principale del programma. Le figure 12, 13, 14 e 15 riassumono le *Classi* utilizzate, ciascuna con i *Campi* e *Metodi* caratteristici.

Brevemente, tutti i *Campi* sono rappresentati dalle posizioni X, Y e dall'etichetta. Tutte le *Classi* qui usate hanno il metodo `.display()`, che ne consente la visualizzazione.

La *Classe* Led ha il *Metodo* `.update()`, il quale, come spiegato, se gli viene passato true accende il LED, mentre lo spegne se riceve false.

La *Classe* Button ha il *Metodo* `.active()`, in-



Fig. 9

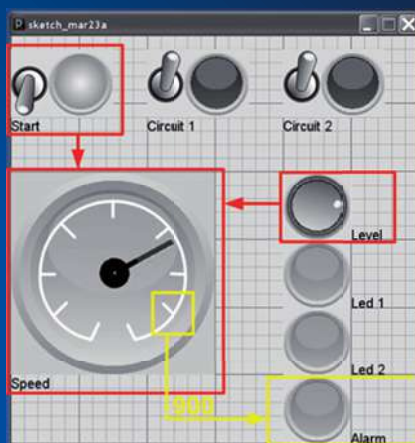


Fig. 10

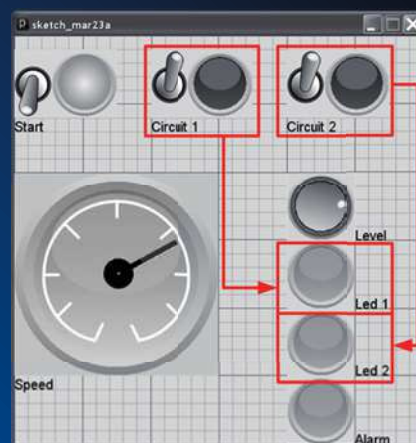


Fig. 11

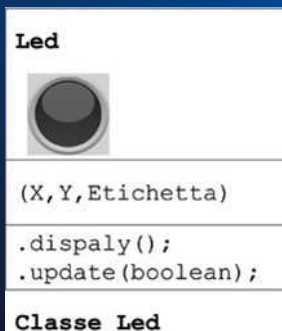


Fig. 12

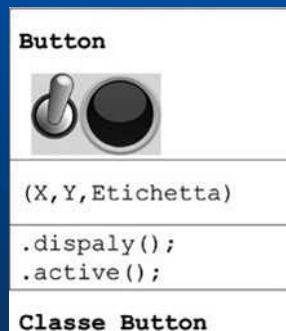


Fig. 13

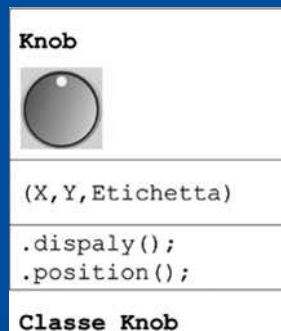


Fig. 14

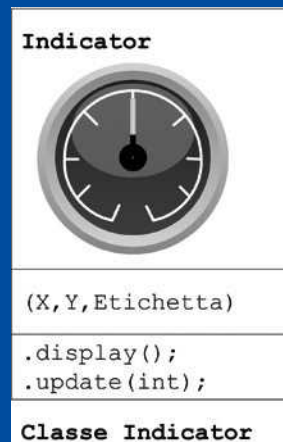


Fig. 15

vece, restituisce un booleano (true/false) a seconda della posizione dell'interruttore. La *Classe Knob* ha il Metodo *.position()*, il quale restituisce un valore intero compreso tra 0 e 1024, proporzionale alla posizione della manopola. La *Classe Indicator* ha il Metodo *.update()* che si aspetta un valore intero compreso tra 0 e 1024; chiaramente il valore 0 corrisponderà all'inizio della scala, mentre 1024 equivarrà alla

posizione di fondo scala. Il codice del programma di interfaccia è contenuto nel Listato 3. Non ci dilunghiamo, in questa lezione, sul codice specifico delle varie *Classi* qui presentate, in quanto è sostanzialmente lo stesso esposto nella terza puntata del nostro corso, ma scritto sotto forma di *Classe*. Quindi, a livello sostanziale non cambia nulla: è solo differente la forma-

Listato 3

```
//definizione degli OGGETTI
Button bt1, bt2, bt3; //definizione degli oggetti della classe Button
Indicator ind1; //definizione degli oggetti della classe Indicator
Knob knb1; //definizione degli oggetti della classe Knob
Led ld1, ld2, ld3; //definizione degli oggetti della classe Led
//-----

//dichiarazione VARIABILI
PImage griglia; //definizione della variabile griglia di tipo PImage
int i=0; //definizione della variabile i che servirà come contatore
//-----

void setup(){
  background(200); //colore di sfondo
  size(600,600); //dimensione dell'area di disegno

  bt1 = new Button(0,20,"Start"); //istanza dell'oggetto bt1 e sua costruzione
  bt2 = new Button(200,20,"Circuit 1"); //istanza dell'oggetto bt2 e sua costruzione
  bt3 = new Button(400,20,"Circuit 2"); //istanza dell'oggetto bt3 e sua costruzione
  ind1 = new Indicator(0,200,"Speed"); //istanza dell'oggetto ind1 e sua costruzione
  knb1 = new Knob(400,200,"Level"); //istanza dell'oggetto knb1 e sua costruzione
  ld1 = new Led(400, 300, "Led 1"); //istanza dell'oggetto ld1 e sua costruzione
  ld2 = new Led(400, 400, "Led 2"); //istanza dell'oggetto ld2 e sua costruzione
  ld3 = new Led(400, 500, "Alarm"); //istanza dell'oggetto ld3 e sua costruzione

  griglia = loadImage("griglia_600_600.png"); //si carica l'immagine di una griglia per facilitare il posizionamento degli elementi dell'interfaccia
}

void draw(){
  image(griglia,0,0); //rappresentazione dell'immagine griglia

  bt1.display(); //rappresentazione di bt1
  bt2.display(); //rappresentazione di bt2
  bt3.display(); //rappresentazione di bt3
  ind1.display(); //rappresentazione di ind1
  knb1.display(); //rappresentazione di knb1
  ld1.display(); //rappresentazione di ld1
  ld2.display(); //rappresentazione di ld2
```

(Continua)

Listato 3 – segue

```

ld3.display(); //rappresentazione di ld3

//funzionamento dell'interfaccia

//se il bt1 "Start" è attivo
//l'indicatore crescerà fino a raggiungere il valore impostato dalla manopola knob1 "Level"
//se riduciamo il valore della manopola, il valore rappresentato dall'indicatore si ridurrà

if(bt1.active()){ //se il l'interruttore bt1 "Start" è in posizione attiva

  println(knob1.position()); //sul terminale viene rappresentato il valore trasmesso dalla posizione della
  manopola knob1

  if(i<knob1.position()){ //se la posizione della manopola è superiore alla posizione dell'indicatore
    i +=1; //incremento la variabile i
    ind1.update(i); // aggiorno la posizione dell'indicatore ad i
  }
  if(i>knob1.position()){ //se la posizione della manopola è inferiore a quella dell'indicatore
    i -=1; //decremento i
    ind1.update(i); //aggiorno la posizione dell'indicatore ad i
  }

  if(i>900){ //se la variabile i oltrepassa 900
    ld3.update(true); //aggiorno lo stato del led3 "Alarm" ad acceso
  }else{
    ld3.update(false); //altrimenti lo stato di led3 è spento
  }

}

if(bt2.active()){ //se l'interruttore bt2 è acceso
  ld1.update(true); //aggiorno lo stato di ld1 a "true", accendo il led
}
else{
  ld1.update(false); //altrimenti lo stato del led è spento
}

if(bt3.active()){ //se l'interruttore bt3 è acceso
  ld2.update(true); //aggiorno lo stato di ld2 a "true", accendo il led
}
else{
  ld2.update(false); //altrimenti lo stato del led è spento
}

}

```

lizzazione, come spiegato in riferimento al codice della *Classe Led*. Comunque vi invitiamo ad analizzare il codice delle varie *Classi* e ad apportare tutte le modifiche che ritenete opportune a seconda delle vostre necessità. Una piccola nota: la Fig. 16 mostra l'interfaccia completa e funzionante, ma dobbiamo notare che in essa è presente una griglia di sfondo, richiamata da:

```
griglia = loadImage("griglia_600_600.png");
```

Praticamente, abbiamo creato un'immagine rappresentante un quadrato di 600x600 pixel, con una griglia interna spaziata di 20 pixel, la quale ci consente di avere un riferimento quando piazziamo i controlli e gli indicatori sull'interfaccia, permettendoci in maniera

più rapida di allinearli. Una volta completata l'interfaccia, basterà eliminare (commentare) la riga di codice precedente, che richiama la griglia, per far sparire questa immagine. ■



Fig. 16